Expressions

Types of expressions

An expression describes a computation and evaluates to a value

\[
\begin{align*}
18 + 69 & \quad 6 \div 23 \\
\sin \pi & \quad \log_2 1024 \\
7 \mod 2 & \quad \sqrt{143161} \\
\sum_{i=1}^{100} i & \quad \lim_{x \to 0} \frac{1}{x}
\end{align*}
\]

Operators and operands are also expressions

So they evaluate to values

Evaluation procedure for call expressions:

1. Evaluate the operator and then the operand subexpressions
2. Apply the function that is the value of the operator
to the arguments that are the values of the operands

Call Expressions in Python

All expressions can use function call notation

(Demo)

Evaluating Nested Expressions

Expression tree

Operand subexpression

Value of the whole expression

Value of subexpression

1st argument to mul

Operator subexpression

Value of the whole expression

Operand subexpression

Value of subexpression

Expression tree
Names, Assignment, and User-Defined Functions

Types of Expressions

Primitive expressions:
- Number or Numeral
- Name
- String

Call expressions:
- max
- 2
- 3

Operator

Operand

Operand

An operand can also be a call expression

Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Environment Diagrams Solution

Assignment Statements

1. Evaluate all expressions to the right of = from left to right.
2. Bind all names to the left of = to those resulting values in the current frame.

Defining Functions
Defining Functions

Assignment is a simple means of abstraction: binds names to values.

Function definition is a more powerful means of abstraction: binds names to expressions.

```
>>> def square(formal parameters):
    return return expression
```

Function body defines the computation performed when the function is applied.

Execution procedure for def statements:
1. Create a function with signature `<name>(<formal parameters>)`
2. Set the body of that function to be everything indented after the first line
3. Bind `<name>` to that function in the current frame.

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):
1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```
from operator import mul
def square(x):
    return mul(x, x)
square(-2)
```

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:
- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I’ll say all day:

An environment is a sequence of frames. A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

- To look up some name in the body of the square function:
  - Look for that name in the local frame.
  - If not found, look for it in the global frame.

      (Built-in names like "max" are in the global frame too, but we don’t draw them in environment diagrams.)

      (Demo)